Loader Handout

This handout is intended to accompany material covered during lectures and is not considered a replacement for lectures. Dissemination of this handout (including posting on a website) is explicitly prohibited unless permission is obtained. Please report any errors to nanaeem@uwaterloo.ca.

1 Challenges for Relocation

We cannot expect that the memory required to load and execute a program will always be available as a contiguous chunk of memory starting at address 0x00. Therefore, a loader must be able to relocate an assembled program to any starting address. The starting address α is supplied to the loader by the OS.

An obstacle during relocation is if the original assembly program used the assembler directive `.word <label>`. For a `.word <label>`, the assembler outputs the 32-bit address corresponding to the location where `label` was declared. Since the assembler assumes that addressing starts at 0, we have a problem when this program is loaded into memory at an address other than 0. The address provided for the `label` will be off by α, the actual starting address.

To fix this error, the relocator must find all locations in the loaded program that originated from a `.word <label>` and add α to the address currently stored at that location.

Since binary programs have no concept of labels, the information of which 32-bit sequence used to be a `.word <label>` is lost in binary. Therefore, this information must be conveyed to the loader (or more precisely the relocator). In CS241 we use the MERL specifications which provide a relocation table with relocation entries specifying each address in the generated binary that originated due to ta `.word <label>` in the assembly program.

2 MERL format

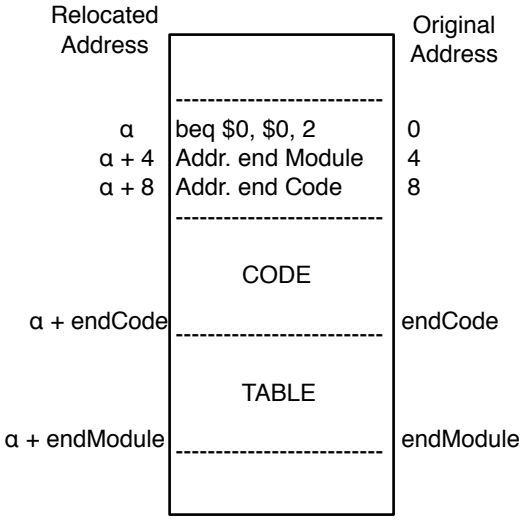
The following example illustrates the MERL file in assembly, the memory addresses for each line when loaded at starting address 0, and the actual 32-bit binary for each line in the assembly file

```
; .merl VERSION OF RELOCATION EXAMPLE
; MERL = MIPS executable relocatable linkable

                                  ; address     machine l.
beq $0, $0, 2 ; skip over header  ; 0x00000000  0x10000002
.word endmodule                   ; 0x00000004  0x0000003c
.word endcode                     ; 0x00000008  0x0000002c

        lis $3                    ; 0x0000000c  0x00001814
        .word 0xabc               ; 0x00000010  0x00000abc
        lis $1                    ; 0x00000014  0x00000814
reloc1: .word A                   ; 0x00000018  0x00000024
        jr $1                     ; 0x0000001c  0x00200008
        B:
        jr $31                    ; 0x00000020  0x03e00008
        A:
        beq $0, $0, B             ; 0x00000024  0x1000fffe
reloc2: .word B                   ; 0x00000028  0x00000020
endcode:

.word 1      ; relocate           ; 0x0000002c  0x00000001
.word reloc1 ; location           ; 0x00000030  0x00000018

.word 1      ; relocate           ; 0x00000034  0x00000001
.word reloc2 ; location           ; 0x00000038  0x00000028
endmodule:
```

3 Relocating Loader

The last column in the example above is the input to a loader which knows how to read the MERL format to relocate the code at any starting address α. The relocating loader first loads the program at starting address α as shown in the diagram below::

(Note: in the diagram below the beq instruction is shown in assembly for clarity. Remember that what is loaded into memory is the binary representation of the program.)



The relocator runs the following code to relocate each relocation entry:

i ← α + MEM[α+8]	The third word in the original code contained the address of the endCode label. Since the program is now loaded at α, the 3rd word is α+8. Therefore we read the memory at this location This gives us the value of endCode in the original code Since the program is loaded at α we add α to this AGAIN!!
end ← α + MEM[α+4]	The 2nd word in memory contains the address of the end of file So we load it through MEM[α+4]. The value at that address was based on addresses starting at 0, so we add α
while (i < end){	This loop takes us through the footer table containing relocation entries
if (MEM[i] == 1) {	Remember that each entry is 2 words. The first word has value ONE to indicate a relocation entry
MEM[α + MEM[i+4]] += α	Since MEM[i] is the .word 1, MEM[i+4] is the address that contained a .word label. Since we started at α, the .word label is actually at α + MEM[i+4]. We READ in that value MEM[α + MEM[i+4]] and this represents what .word label would have had we started at 0. Add α to this and store it back
i +=8;	Jump to next table entry
}	
}	

4 Tools

4.1 Relocatable Assembler: cs241.relasm

- cs241.binasm does not know how to create MERL output.
- cs241.relasm is an assembler which will create MERL output.

4.2 Relocation Tool: cs241.merl

- Input: merl file and relocation address
- Output: NON-relocatable mips file, i.e., MERL header and footer has been removed. Ready to load at given address

4.3 Simulators: mips.twoints and mips.array

- The simulators CAN take a second argument: an address at which to load a mips file

4.4 Example

```
java cs241.relasm < input.asm > output.merl
//generates a relocatable object file
//(which can run at address 0 or be relocated)

java cs241.merl 0x12345678 < output.merl > output.mips
//generates a NON-relocatable binary file
//(can only be run at address 0x12345678)

java mips.twoints output.mips 0x12345678
```